Dialog Handler

Overview

The dialog handler is a used to create dialog that support basic branching behavior. If you want to do straight forward dialogs that have no branching at all, you might as well just call the Voice Handler directly. But if there are some branching, even simple like breaking the dialog if the player stop looking at a character, then the Dialog Handler is what you want.

Basic functionality

The dialog is setup just before it is a about to get played, so there is not special init (other than what you do for the voice handler). You begin by calling Dialog_Begin() which makes the initial preparations and allows you to call the other functions. After that it is time to add a branch using Dialog_AddBranch. You can then use Dialog_AddSubject to add voice subjects to that branch. Once you add a new branch, then those subjects will belong to the newly added branch. When all branches and subjects are added, call Dialog_End to run the dialog. Dialog_End will also check which characters are used in this dialog, and by using Dialog_CharacterIsActive you can check which are currently in the middle of a dialog.

Example:

```
Dialog_Begin();
    Dialog_AddBranch("A");
        Dialog_AddSubject("AdamEnter1");
        Dialog_AddSubject("AdamEnter2");
    Dialog_AddBranch("B");
        Dialog_AddSubject("AdamLeave");
Dialog_End();
```

This will create two branches, “A” and “B”. The first contains “AdamEnter1” and “AdamEnter2”, and the other “AdamnLeave”. When Dialog_End() is called, “A” will begin playing, starting with “AdamEnter1” as it was the first added. If you want to start with some other branch, just supply its name as a parameter for Dialog_End(), like this:

```
Dialog_End("B")
```

This would start with branch “B” instead.

Also of interest is that you can make one branch go directly to another by supplying the next dialog as a second parameter, like this:

```
Dialog_AddBranch("A", "B")
```

Now when “A” is done, “B” is started. This might seem a bit pointless at first, but is really useful when you add branching events (more on that soon).

Branch Events

Overview

These events are needed in order to get the branching behaviors you are after. Events are added after having added subject and will belong that subject. They can in two types `LineEvents` and `EndEvent`. The first type is checked after each voice line has completed playing. The second is checked when the entire subject is over.

The existing functions carry this syntax:
`Dialog_Add[type]_[condition]` *(more on conditions in a few lines)*
Some examples for functions:

```
void Dialog_AddLineEvent_Callback(const tString&in asCallbackFunc, const
tString&in asNewBranch)
void Dialog_AddEndEvent_PlayerNotLookingOrOutOfRange(const tString&in
asNewBranch)
```

Note that if asNewBranch is “”, then the dialog does not jump to a new branch but just stops.

Conditions

The different conditions available are:

OutOfRange
This is true if the player is out of range, as set in Voice_SetSource(..) with the PlayerListeningRange parameter. This applies to all characters in the dialog with a proper source entity and is only true if all of them are out of range.

PlayerNotLooking
This is true when the player is no longer looking at the source entity set with Voice_SetSource(..). This applies to all characters in the dialog with a proper source entity and is only true if all of them are not looked at.

Variables
The functions for this these are: SetVar, IncVar, VarIsSet, VarEquals, VarGreater and VarLesser.
This is a very simple variable system where each variables carries an integer value. It is very nice to use to check if some information has been mentioned, by simply doing SetVar(“Info”) and then deciding on branch depending on if some info is set or not.
Currently variables only support the type `EndEvent`.
Note that setting variables never results in a branching, so make sure to declare these before any branching event.

Callback
This is a custom callback that can be used to let what ever code determine if there should be a branch or not. The syntax is:
`bool MyFunc(const tString&in asBranch, const tString&in asBranchSubject, int alLineIndex, const tString&in asNewBranch)`

asNewBranch is very handy when you do not want too many callbacks function to provide similar functionality. For example if the branch should be determined by the name of a cat, then you can do a function like this:

```
bool MyFunc(const tString&in asBranch, const tString&in asBranchSubject, int
alLineIndex, const tString&in asNewBranch)
{
  return asNewBranch == msSomeSavedVariableOfCatName;
}
```

The declared events can then look like this:

```
Dialog_AddLineEvent_Callback("MyFunc", "Snoopy");
Dialog_AddLineEvent_Callback("MyFunc", "Shaggy");
```

Example

Now for some full test source. Lets say that we have a dialog that should stop playing whenever the player stops looking at the character, then we do like this:

```
Dialog_Begin();
    Dialog_AddBranch("A");
        Dialog_AddSubject("AdamEnter1-2");
            Dialog_AddEndEvent_PlayerNotLooking("B");
    Dialog_AddBranch("B");
        Dialog_AddSubject("AdamLeave");
Dialog_End();
```

Response selection

Overview

The dialog system allow the player to choose different option that will determine how the outcome of a dialog. After a subject has finished playing the DialogHandler will send a message to put in the game in “Response Mode” (more information on this at the end of this section) and the player will have to make a choice before the dialog continues. Each of the choices are called `ResponseOptions` and it is possible to set up conditions for if they should be present nor not, as well as declare events that happen if the ResponseOption is chosen.

ResponseOptions are added after having added a subject and uses the function Dialog_AddResponseOption(ebtry). It looks like this:

```
Dialog_AddBranchAndSubject("A");
  Dialog_AddResponseOption("TextEntry1","NewBranch1");
  Dialog_AddResponseOption("TextEntry2","NewBranch2");
  ...
```

This code will give the player two choices and jump to either “NewBranch1” or “NewBranch2” depending on their choice. The first argument for each option, “TextEntry1” and “TextEntry2”, is an entry in the lang file. The category will always be the name of the level (the file name without extension). The second argument is the branch to jump to if the option is chosen. If this is “”, the dialog is ended.

In order to make sure that the code looks good a certain syntax for the Entry should be used and is as follows:

`Response_[Scene]_[Subject]_[Text]`
Scene: The scene as defined in the voice handler.
Subject: The name of the subject that the response selection follows.
Text: Something that summarizes actual text for this choice.
Example:

```
Dialog_AddResponseOption("Response_FredAndDanny_AskIfWantHug_YesPlease",
"");
```

Note: In the examples here I am not using correct syntax, but that is only to keep the text compact!

ResponesOptions and BranchEvents

As noted above, `BranchEvents` can also be added after a subject, and if there are both `ResponseOptions` and `BranchEvents` after a subject, the BranchEvents are checked first and then the ResponseOptions are handled. So for instance:

```
Dialog_AddBranchAndSubject("A");
  Dialog_AddResponseOption("TextEntry1","NewBranch1");
  Dialog_AddResponseOption("TextEntry2","NewBranch2");
  Dialog_AddEndEvent_VarIsSet("Var", "NewBranch3")
...
```

In the code above, after having played Subject “A”, the dialog will jump to “NewBranch3” if the variable “Var” has been set. It is only if the the VarIsSet-test fails that the Response selection is started.

Variables

It is also possible to check and set internal dialog variables at each ResponseOption. These are the same variables that are checked and set by the BranchEvents. The functions for doing this are:
`Dialog_AddResponseCondition_*`
This makes check and see if a ResponseOption should be used. * is VarIsSet, VarNotSet, VarEqual, VarLesser or VarGreater.
`Dialog_AddResponseEvent_*`:
This sets a variable if the ResponseOption is chosen. * is SetVar or IncVar.

These shall be set after a response option has been added. For example: subject, the BranchEvents

are checked first and then the ResponseOptions are handled. So for instance:

```
Dialog_AddResponseOption("TextEntry1","NewBranch1");
  Dialog_AddResponseCondition_VarEqual("BodyCount", 3);
  Dialog_AddResponseEvent_SetVar("Foo");
```

This will add the response “TextEntry1” if BodyCount has a value of 3. If this option is chosen, then the variable “Foo” is set.

Tip::
If you have Response Selection that pops up many times (for instance some topic selection) then an easy way to make an option only appear once is to use Dialog_AddResponse_OneTimeCheck(...). This adds a condition to check if a var is not set, and if chosen sets that variable.

Time limit and Default Options

It is possible to set a time limit to a Response Selection by using Dialog_SetResponseTimeLimit(...), where 0=means unlimited amount. If time runs out the default ResponseOption will be chosen. This is simple an option with an an empty entry (named “” that is). This option must always be declared after all other ResponseOptions have been declared. The default option will never be displayed.

If a Response selection only has a single visible option (meaning no default options counted), then that option will automatically be selected. If there is only a default option then that will be selected automatically as well.

Selection implementation

The Dialog Handler does not implement any GUI or input for the actual selection, instead this is up to the user implementation. When a Response Selection is encountered, the Dialog handler will call `_Global_StartResponseSelection` in the current player state. If none is find, the selection is skipped. If found then it will send the following arguments:

0	float with time limit.
1	The number of visible options.
2,3,..	The text entries for each option as a String

When the selection is done, the implementation must call ReturnResponseSelectChoice in cLuxDialogHandler with the selected option. If time has run out, then -1 shall be sent as argument.

Locking the player into a conversation

There are a few cases in SOMA where we lock the player into a conversation i.e. they can't move their feet while it's happening.

`Dialog_StartConversation` is the simplest form of this; it just locks the player where they're standing right now. `Dialog_StartConversation_PosBased` actually moves the player's viewpoint to a specified position before starting the conversation, which is useful if you're worried about

eyelines etc.